

WebServices:

- WebServices are the Custom Business Logic Classes, Which can be accessible from the Third Party Applications.
- (Ex: From Banking Applications, Finance Applications, Insurance Applications, SAP Applications,...etc.)
- By using WebServices, we can expose the Business Logics, Data, and Functionalities to the Third party Applications.
- WebServices will make the Multiple Disparate Systems can communicate with other, in-order to exchange the data / business logic / functionalities.
- WebServices are Purely Platform Independent and Language Independent.

We have the below 2 types of WebServices.

1. REST Based WebServices (RestFul)
2. SOAP Based WebServices.

REST Based WebServices:

REST --> Representational State Transfer.

REST is an Architectural Style, Which has been purely implemented based on the HTTP Protocol.

It is using the Old HTTP Protocol to exchange the data / functionalities / business logic between the disparate systems.

They are using the old HTTP Protocol, and added a set of enhancements on Top of HTTP Protocol to exchange the data between the multiple Third Party Applications.

In RestFul WebServices, each Business Logic/ Functionality / Data will be referred as a "Resource".

For Each resource, we can provide the resource name. We can access each resource by using the "URI (Uniform Resource Identifier)".

Syntax: <http://www.sample.com/<resourceName>>;

Salesforce provides a resource name for each object as below. Which is also referred as "Object ID / Schema ID / Object Key Prefix".

Object Name	Resource Name / Object ID
Account	---> 001
Contact	---> 003
Lead	---> 00Q
Campaign	---> 701
Case	---> 500
Solution	---> 501

We can access each object records as below..

Ex: <https://ap16.salesforce.com/001> --> Account Records

<https://ap16.salesforce.com/005> --> User Records

<https://ap16.salesforce.com/801> --> Order records

To access the resources over the HTTP Protocol, we have to use the below HTTP Methods.

HTTP Protocol provides the below 5 Methods to be used to interact with the Resources to perform the operations.

1. GET : Used to Get the Records from the resource
2. DELETE : Used to Remove the records from the Resource
3. POST : Used to insert the New Records into the Resource
4. PUT : Used to Update the Existing records in the resource

5. PATCH : Used to Perform the Upsert Operation on the records. (i.e. Update + Insert)

Note:

RESTful WebService will expose the response in the form of either "JSON / XML / TEXT".

Most of the Mobile Applications, Cloud Applications, Web Applications are supporting the Restful WebServices". Because Restful WebService are very lightweight, easy to implement, and easy to parse.

By using Apex Programming, we can build our own Restful Webservice and can expose the WebService to the Third Party Systems.

To Prepare the Restful WebServices, Apex provides the below Annotations.

1. `@RestResource()`
2. `@HttpGet()`
3. `@HttpDelete()`
4. `@HttpPost()`
5. `@HttpPatch()`
6. `@HttpPut()`

@RestResource():

This annotation is used to expose a Business Logic Class to the Third Party System. So that it can consume the WebService, to exchange the data.

Note:

Each WebService Class should be defined with the "Global" access specifier.

Each WebService Class should be identified by using a "Resource Name", which can be provided by using the "`@RestResource()`" annotation.

i.e. Each Restful WebService class should be pre-fixed with "@RestResource()" annotation.

Syntax:

```
@RestResource(URLMapping='/<ResourceName>/*)'

Global Class <ClassName>

{

    // Write the Business Logic..

}
```

Note: URLMapping parameter is used to specify the resource name, through which the

WebService class can be accessible.

Ex:

```
@RestResource(URLMapping='/AccountsService/*')

Global Class AccountsManager

{

    // Write the Business Logic..

}
```

Ex:

```
@RestResource(URLMapping='/CaseManagerService/*')

Global Class CaseHelper

{

    // Write the Business Logic..
```

```
}
```

Accessing the RestFul WebService: (EndPoint URL)

Each RestFul WebService can be accessible by the Third Party Systems with the help of "URI" as below..

Syntax:

```
https://<ServerNumber>.Salesforce.com/Services/Apexrest/<ResourceName>
```

Ex: https://ap16.salesforce.com/services/apexrest/AccountsService

```
https://ap16.salesforce.com/services/apexrest/CaseManagerService
```

@HttpGet() Method:

HTTPGet Method is used to get required resources / information / data from the Provider / Server.

This annotation can be applicable only on the Method Level.

Note: All the WebService Methods should be defined with "Static" keyword and should be defined with "Global" access specifier.

Syntax:

```
@HttpGet()
```

```
Global static <ReturnType> <MethodName>(<Parameters List>)
```

```
{
```

```
// Write the Business Logic..
```

```
}
```

Note:

We can't define Two Methods inside the WebService Class with the Same Annotation. (i.e. We can't define Two Methods with the "HTTPGet()" annotation)

Ex:

```
@RestResource(URLMapping='/AccountsService/*')

Global Class AccountsHelper

{

    @HttpGet()

    Global static List<Account> GetAllAccounts()

    {

        // Write the Business Logic, to Get All Account Records..

    }

}
```

Accessing the WebService:

Ex: <https://ap16.salesforce.com/services/apexrest/AccountsService>

We can invoke the WebService from the below tools.

1. Workbench Tool.
2. Postman Tool
3. Through Apex Programming
4. By using Integration Tools (Ex: Mulesoft, Tibco, Webmethods,...etc)
5. By using Third Party Applications.

UseCase 1:

Create a Restful WebService to get all the Account Records exist inside the object.

Service Class:

```
@RestResource(URLMapping='/AccountsService/*')

Global class AccountRecordsService {

    @HttpGet()

    Global static List<Account> GetAllAccounts()  {

        List<Account> lstAccounts = [Select id, name, rating, annualrevenue, phone,
fax,type, ownership, customerpriority__c, active__c from Account Order by
name];

        return lstAccounts;
    }

}
```

EndPoint URL:

<https://ap16.salesforce.com/services/apexrest/AccountsService>

Method Name: GET

UseCase 2:

Create a Restful WebService to expose all the Hiring Manager Records, which are associated with "Bangalore Location".

Service Class:

```
@RestResource(URLMapping='/HiringManagerService/*')

Global class HiringManagerRecordsService {

    @HttpGet()

    Global static List<Hiring_Manager__c> GetBangaloreHRRecords()  {
```

```
        List<Hiring_Manager__C> lstHRRecords = [Select id, name,  
location_name__C, email_id__C, contact_number__C from  
Hiring_Manager__C Where location_name__C = 'Bangalore'];  
  
        return lstHRRecords;  
  
    }  
  
}
```

EndPoint URL:

<https://ap16.salesforce.com/services/apexrest/HiringManagerService>

Method Name: GET

RestContext Class:

RestContext Class will provides a communication mechanism between the Consumer and the Provider. So, that Consumer can provide the input Parameters to the Provider. Then provider will collect the required data based on the input parameters and will send the response back to the Consumer.

It contains 2 inner classes as below.

1. RestRequest Class.
2. RestResponse Class.

RestRequest Class:

This Class is used to make a request to the Restful WebService by passing the required input parameters.

Note:

It will store the Input Parameters in the form of "Key-Value" pair Collection. Where Key is the Parameter Name, and Value is the Parameter Value.

Property:

Params:

This Property Contains all the Input Parameters in the form of Key-Value pair collection.

WebService will collect the input value from the RestRequest and get the required matching Data based on the input values.

Syntax:

```
Map<String, String> <collectionName> = RestContext.request.Params;
```

RestResponse Class:

This Class is used to collect the response from the WebService.

UseCase 3:

Create a Restful WebService to Get the Case Details based on the Supplied Case Number at runtime.

Class Code:

```
@RestResource(URLMapping='/CaseDetailsService/*')

Global class CaseManagerHelper {

    @HttpGet()

    Global Static Case GetCaseRecordByNumber()  {

        Map<String, String> inputParams = RestContext.request.Params;

        Case caseRecord = [Select id, caseNumber, status, type, priority,
origin,Reason, Subject, Description from Case
Where CaseNumber =: inputParams.Get('cNumber')];

        return caseRecord;
    }
}
```

}

Testing:

URL : /services/apexrest/CaseDetailsService/?cNumber=00001016

Method: GET

UseCase 4:

Create a RestFul WebService, to Get the All the Customers Information from the Object based on the specified "Industry Name" at runtime.

Class Code:

```
@RestResource(URLMapping='/SearchAccountsService/*')

Global class AccountsSearchHelper {

    @HttpGet()

    Global static List<Account> GetAccountsByIndustry()  {

        Map<String,String> inputParams = RestContext.request.Params;

        List<Account> lstAccounts = [Select id, name, rating, industry, annualrevenue,
                                      phone, fax, type, customerPriority__C, active__C
                                      from Account
                                      Where industry =: inputParams.Get('indName')];

        return lstAccounts;
    }
}
```

Testing Process:

URL: /services/apexrest/SearchAccountsService/?indName=Education

Method: GET

Assignments:

1. Create a Restful WebService, to Fetch All the Cases based on the Specified Priority.
2. Create a Restful WebService, to fetch all the Hiring Managers based on the specified Location Name.
3. Create a Restful WebService, to fetch all the Lead Records, based on the specified Status.
4. Create a Restful WebService, to Fetch all the Opportunities based on the Specified Account Name.
5. Create a Restful WebService, to fetch all the Related Positions based on the Specified Hiring Manager Name.

HttpDelete Method:

By using this method we can fetch the related records from the object based on the supplied input parameters and we can remove the records from the object. We can remove either one or more records from the object, by using this method.

Syntax:

```
@HttpDelete()
```

```
Global static <ReturnType> <MethodName>()  
{  
    // Write the Business Logic to remove the records from the object.  
}
```

UseCase 5:

Create a Restful WebService, to remove the Account Records from the object based on the supplied Account Name at runtime.

Class Code:

```
@RestResource(URLMapping='/SearchAccountsService/*')  
  
Global class AccountsSearchHelper {  
  
    @HttpDelete()  
  
    Global Static string DeleteAccountsByName()  {  
  
        Map<String, String> mapInputs = RestContext.request.Params;  
  
        List<Account> lstAccounts = [Select id, name, rating, industry from  
Account  
  
                                         Where name =:  
mapInputs.Get('accName')];  
  
        if(! lstAccounts.isEmpty()) {  
  
            Delete lstAccounts;  
  
            return 'Records Deleted.';  
  
        }  
  
        else  
  
            return 'No Matching Records Found.';
```

```
    }  
}
```

Testing Process:

URL: /services/apexrest/SearchAccountsService/?accName=Sampath

Method : DELETE

Assignments:

1. Create a Restful WebService, to Remove the Hiring Manager based on the Specified

Name.

2. Create a Restful WebService, to Remove all the Related Cases from the Object

based on the specified Account Name.

3. Create a Restful WebService to remove all the Related Contacts based on the

Account Name.

HTTPPost Method:

By using this Method, we can insert one or more records into the Objects. Upon invoking the request, we have to supply the required record details to the WebService in the form of "JSON Format".

We have to embed the request details inside the Request Body.

Syntax:

```
@HttpPost()
```

```
Global static <ReturnType> <MethodName>([Parameters])
```

```
{  
    // Write the Business Logic..  
}
```

UseCase 6:

Create a RestFul WebService, to Insert a Contact Record inside the Object. Supply the required Field values in the form of "JSON Format".

Service Class:

```
@RestResource(URLMapping='/ContactRecordsService/*')  
  
Global class ContactRecordsHelper {  
  
    @HttpPost()  
  
    Global static String CreateNewContact(string fName, string lName, string cTitle,  
    string cEmail, string cPhone, string cFax, string cMobile, string cCity, string cState,  
    string cCountry) {  
  
        Contact con = new Contact();  
  
        con.FirstName = fName;  
  
        con.LastName = lName;  
  
        con.Title = cTitle;  
  
        con.Email = cEmail;  
  
        con.Phone = cPhone;  
  
        con.Fax = cFax;  
  
        con.MobilePhone = cMobile;  
  
        con.MailingCity = cCity;  
  
        con.MailingState = cState;
```

```
con.MailingCountry = cCountry;

insert con;

if(con.Id != null)

{

    return 'Contact Record Created with ID..: '+ con.Id;

}

else

    return 'Contact Record Creation has been Failed.';

}

}
```

Testing Process:

URL: /services/apexrest/ContactRecordsService

Method: POST

Request Body:

```
{

    "fName":"Ram",

    "lName":"Kumar",

    "cEmail":"ramkumar@gmail.com",

    "cFax":"9900887788",

    "cPhone":"9900990000",

    "cMobile":"9988998899",

    "cTitle":"Product Manager",
```

```
        "cCity":"Hyderabad",  
        "cState":"Telangana",  
        "cCountry":"India"  
    }  

```

UseCase 7:

Create a RestFul WebService, to Insert a Hiring Manager Record inside the Object.

Service Class:

```
@RestResource(URLMapping='/HiringManagerRecordService/*')  
  
Global class HiringManagerRecordsHelper {  
  
    @HttpPost()  
  
        Global Static String CreateHRRecord(string hrName, string hrEmail, string  
        hrPhone, string hrCity)  {  
  
            Hiring_Manager__C hrRecord = new Hiring_Manager__C();  
  
            hrRecord.Name = hrName;  
  
            hrRecord.Email_ID__c = hrEmail;  
  
            hrRecord.Contact_Number__c = hrPhone;  
  
            hrRecord.Location_Name__c = hrCity;  
  
            insert hrRecord;  
  
            if(hrRecord.Id != null)  
  
                return 'HR Record Inserted Successfully.';  
  
            else  
  
                return 'HR Record Insertion Failed.';  
        }  
}
```

```
    }  
}  
}
```

Testing Process:

URL: /services/apexrest/HiringManagerRecordService

Method Name: POST

Request Body:

```
{  
    "hrName": "Shubha Raghynadhan",  
    "hrCity": "Bangalore",  
    "hrEmail": "shubha@gmail.com",  
    "hrPhone": "9900990000"  
}
```

Assignments:

1. Create a RestFul WebService, to Insert a Lead Record inside the Object.
2. Create a RestFul WebService, to Insert an Opportunity Record inside the Object.
3. Create a RestFul WebService, to Insert a Position Record inside the Object.

UseCase 8:

Create a RestFul WebService, to Insert 2 Lead Records inside the Object.

Service Class:

```
@RestResource(URLMapping='/BulkLeadService/*')
```

```
Global class LeadRecordsHandler {  
    @HttpPost()  
  
    Global static String CreateBulkLeadRecords()  {  
        string leadRecordDetails = RestContext.request.requestbody.ToString();  
  
        List<Lead> leadRecordsToInsert = (List<Lead>)  
System.JSON.deserialize(leadRecordDetails, List<Lead>.Class);  
  
        if(! leadRecordsToInsert.isEmpty())  
  
        {  
            insert leadRecordsToInsert;  
  
            return 'Lead Records Inserted';  
  
        }  
  
        else  
  
            return 'Lead Records Not Found. Invalid Data.';  
    }  
}
```

Testing Process:

URL : /services/apexrest/BulkLeadService

Method : POST

Request Body:

```
[  
  {  
    "FirstName":"Praveen",
```

```
        "LastName":"Kumar",
        "Title":"Sales Head",
        "LeadSource":"Web",
        "Rating":"Hot",
        "Industry":"Banking",
        "AnnualRevenue":4500000,
        "Phone":"9900889988",
        "Fax":"8899887766",
        "MobilePhone":"9900334433",
        "Company":"TCS Inc.",
        "Status":"Open - Not Contacted",
        "City":"Hyderabad",
        "State":"Telangana",
        "Country":"India"
    },
    {
        "FirstName":"Balaram",
        "LastName":"Kumar",
        "Title":"Project Manager",
        "LeadSource":"Web",
        "Rating":"Wrm",
        "Industry":"Technology",
```

```
        "AnnualRevenue":3700000,  
        "Phone":"7788776655",  
        "Fax":"9988997755",  
        "MobilePhone":"8899665533",  
        "Company":"IBM Inc.",  
        "Status":"Open - Not Contacted",  
        "City":"Bangalore",  
        "State":"Karnataka",  
        "Country":"India"  
    }  
]
```

@HTTPPUT Method:

By using this feature, we can update either one or more existing records inside the object by passing the values for the required fields.

Note:

Upon updating the records, we have to supply the "RecordID".

Note:

While Updating the records, we have to pass the required information to the WebService in the form of "JSON Format" through request body.

Syntax:

```
@HttpPut()
```

```

Global Static <ReturnType> <MethodName>(<Parameters>)

{
    // Write the Code to update the records.

}

```

UseCase 9:

Create a RestFul WebService, to Update the Case Record based on the Case Number, by assigning the values as below.

```

Case:Status = 'Working'

Case:Priority = 'High'

Case:Reason = 'Performance'

```

Service Class:

```

@RestResource(URLMapping='/CaseUpdateService/*')

Global class CaseRecordsHelper {

    @HttpPut

    Global static String UpdateCaseRecord(string cNumber, string cStatus, string
    cReason, string cPriority)  {

        Case csRecord = [Select id, caseNumber, status, priority, Reason      from
        Case
                                         Where caseNumber =:
        cNumber];

        if(csRecord.Id != null)

        {

```

```
        csRecord.Status = cStatus;  
  
        csRecord.Reason = cReason;  
  
        csRecord.Priority = cPriority;  
  
        Update csRecord;  
  
        return 'Case Record Updated Successfully.';  
  
    }  
  
    else  
  
        return 'Case Record Not Found. Invalid Case Number.';  
  
    }  
  
}
```

Testing Process:

URL : /services/apexrest/CaseUpdateService

Method: PUT

Request Body:

```
{  
  
    "cNumber":"00001002",  
  
    "cStatus":"Working",  
  
    "cPriority":"High",  
  
    "cReason":"Performance"  
  
}
```

@HTTPPatch Method:

By using this method, we can perform the Update Operation and Insert Operation inside the object at a time.

Note:

While Performing the Patch Operation (i.e. UPSERT), we have to supply the records inside a Collection.

The Records which are having the record Id's those will get updated into the object. And the records which are not having the id's will get inserted as new records inside the object.

Syntax:

```
@HttpPatch  
Global Static <ReturnType> <MethodName>(<Parameters>)  
{  
    // Write the Logic to Insert the Records.  
    // Write the Logic to Update the records.  
}
```

UseCase 10:

Create a Restful WebService, to Update the Hiring Manager Records based on the specified Record ID. And Create a New Hiring Manager Record inside the Object.

Service Class:

```
@RestResource(URLMapping='/HiringManagerUpsertService/*')  
Global class HiringManagerRecordsUtility {  
    @HttpPatch()  
        Global Static string UpsertHRRecords(string recordID, string hrPhone, string  
        hrCity) {
```

```
if(recordID != null && recordID != "") {  
  
    List<Hiring_Manager__C> lstHRRecords = new List<Hiring_Manager__C>();  
  
    // Update the Hiring Manager Record...  
  
    Hiring_Manager__C hrRecord = [Select id, name, location_name__c,  
    contact_number__c      from Hiring_Manager__C      Where id =: recordID];  
  
    if(hrRecord.id != null) {  
  
        hrRecord.Location_Name__c = hrCity;  
  
        hrRecord.Contact_Number__c = hrPhone;  
  
        // Add the record to collection..  
  
        lstHRRecords.Add(hrRecord);  
  
    }  
  
    // Create a New Hiring Manager Record..  
  
    Hiring_Manager__C hr = new Hiring_Manager__C();  
  
    hr.Name = 'Balaram Prasad';  
  
    hr.Location_Name__c = 'Delhi';  
  
    hr.Contact_Number__c = '9900887766';  
  
    hr.Email_ID__c = 'balaram@gmail.com';  
  
    // Add the record to collection..  
  
    lstHRRecords.Add(hr);  
  
    if(! lstHRRecords.isEmpty()) {  
  
        Upsert lstHRRecords;  
  
        return 'Hiring Manager Records Upserted Successfully.';  
    }  
}
```

```
    }

    else

        return 'Upsert Operation Failed.';

    }

else

    return 'Please Provide the Record ID.';

}

}
```

Testing Process:

URL : /services/apexrest/HiringManagerUpsertService

Method : PATCH

Request Body:

```
{

    "recordID":"a032w000008JfCf",

    "hrPhone":"6666000088",

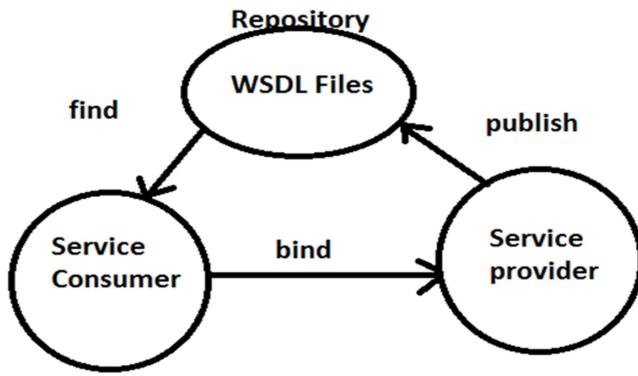
    "hrCity":"Mumbai"

}
```

SOAP - API(Simple object access protocol)

SOA: Service Oriented Architecture

SOA Design Format:-



- It is abbreviated as protocol because it has its own processing roles and nameSpace.
- These client and web services are developed in different languages also. In this case data types and memory sizes of both languages will mismatch.
- So requests and responses are transferred in the form of soap messages.
- At client side ,a stub creates a soap request message with help of binding classes at client side . Similarly , a soap response message will be created by the soap process with the help of binding classes on the server side.
- A soap message contains two parts.
 1. Header
 2. body
- A soap message is called an envelope.
- Header part is optional and the body part is mandatory.

Soap Message

```

<soap:Envelope xmlns:soap="http://Schemas.XmlSoap.org/soap/">
  <soap:Header>
  _____

```

```
_____  
    </soap:Header>  
  
    _____  
  
    <soap:Body>  
  
    _____  
  
    _____  
  
    </soap:Body>  
  
</soap:Envelope>
```

- When a client calls a method of web service then a method call will be converted into xml and it will be inserted in a body part of the soap request message.
- One method call to another method call , xml will be different and I/P values are different in a soap body. So xml inserted in the soap body is called .
- According to the message exchanging format defined in the wsdl file.

rpc/literal document/literal

rpc/encoded document/encoded

- **Rpc style** : Method name used as root tag in xml under soap body.
- **Document style** : Message name used as root tag in xml under soap body.
- **Literal use**: only values are transferred in the xml file.
- **Encoded use**: only values with type are transferred.

rpc/literal:

```
    <soap:body>  
        <hello>  
            <arg>ashok</arg>  
        </hello>  
    </soap:body>
```

rpc/encoded:

```
    <soap:body>
```

```

<hello>
    <arg type="xs:string">ashok</arg>
</hello>
</soap:body>
document/literal:
<soap:body>
    <xs:helloRequest
        xmlns:xs="http://www.ias.com/Schema/types">
        <txt>ashok</txt>
    </xs:helloRequest>
</soap:body>
document/encoded:
<soap:body>
    <xs:helloRequest
        xmlns:xs="http://www.ias.com/Schema/types">
        <txt type="xs:string">ashok</txt>
    </xs:helloRequest>
</soap:body>

```

Soap Exception Handling

- When a client calls operations of web service, if any exception occurs in the operation of web service then directly that exception will not be thrown back to client application because client and server application may not be in the same language.
- If an exception occurred in web service then in binding classes at server side will convert that exception in `<soap:fault>` and fault message inserted in `<soap:body>` response and finally response will be sent back to client application.
- At client side the binding classes convert a `<soap:fault>` into an exception of the client side language and it will be thrown to the client application.
- `<soap:body>` message contains four parts.
 1. `<fault code>`
 2. `<fault string>`
 3. `<fault actor>`
 4. `<details>`

```

<soap:Envelope>
  <soap:body>
    <soap:fault>
      <fault code>xxx</fault code>
      <fault string>xxx</fault string>
      <fault actor>xxx</fault actor>
      <details>xxx</details>
    </soap:fault>
  </soap:body>
</soap:Envelope>

```

- The body part of soap response can have either return value of ws operation or **soap:fault** message.
- If ws operation is successfully executed then the body soap response contains return value. Otherwise it contains **soap:fault** messages.
- A **<fault code>** can be soap:client or soap:server or soap:version mismatch.
- If there is invalid i/p value sent by the client then **<fault code>** is soap:client. If an exception occurs in ws the **<fault code>** soap:server. If the version is mismatched the **<fault code>** is soap:version.
- **<fault string>** is readable message of exception then **<fault action>** is url of ws.
- **<detail>** contains exception object throws by ws in the form of xml.

WSDL(Webservice descriptive language)

SOAP-API also allows you to Salesforce provide two different SOAP API WSDLs (WSDL: Web service description language).

1. Enterprise WSDL
2. Partner WSDL

Enterprise WSDL(Web service description language)

- Enterprise WSDL is a strongly typed WSDL for customers
- It Changes if modifications are made to an Organization Salesforce Configuration.
- It is primarily for Customers.

Partner WSDL(Web service description language)

- Partner WSDL is a loosely typed WSDL for customers
- It is used for any Configuration of Salesforce
- Static and does not change if modification is made to an organization Salesforce Configuration
- It is Primarily for partners

About wsdl file

- Web Service can be developed in one language and its client can be developed in another language.
- In order to send information about webservice about methods in web service,i/p,o/p and location of a web service to the client , A WSDL file will be generated.
- Wsdl file contains five functions
 1. Types section
 2. Message section
 3. Port section
 4. Binding section
 5. Service section
- All the tag attributes that are required for construction of a wsdl file are given by WS-I organization under one nameSpace <http://schema.XmlSoap.org/wsdl/>.
- When constructing a wsdl file the above namespace will be imported into a WSDL file using xmlns keyword.

Types section

This section contains xml schema.

- In this optional either for I/P parameter or return types of methods
- If any equivalent complexType or simpleTypes are created under schema inserted in type section.
- Schema can be defined where as inline/outline means schema can be defined at outline of WSDL file.
- In the type section to define the schema, we need XmlSchema nameSpace to be imported.

- XmlSchema nameSpace imported with type section than the elements of this namespace can only be used under type section, not in wsdl file. In the wsdl file xmlSchema nameSpace will be imported under the root tag.
- In these elements,complexType and simpleType that are created will be stored in a target namespace.
- The elements of targetNamespace are required for the message section so targetNameSpace is also imported with a root tag with prefix “tns”.

```

<definitions xmlns="http://schemas.XmlSoap.org/wsdl"
              xmlns="http://www.w3.org/2001/XMLSchema"
              xmlns:tns="http://www.ias.com/Schema/types"
              targetNameSpace="http://www.ias.com/Schema/types"
              >
    <types>
        <xs:Schema
            targetNamespace="http://www.ias.com/Schema/types"
            >
            _____
            _____
            </xs:schema>
        </types>
    </definitions>

```

Message section

- In this section two messages are created for each method of web service.
- A service provider will tell a client about how many parameters should be sent in a soap request message when calling a webservice method and what type of o/p will come in the soap response.
- A message contains one or more parts where each part indicates a parameter.
- A message section can have multiple messages and each identified with a name.

Example: class

```

Public interface Calculator{
    Int add(int a, int b);
}

Public class Calculate implements Calculator{
    Public int add(int a, int b){
        return a+b;
    }
}

```

```

        }
    }

Sample.wsdl
<definitions xmlns="http://schemas.XmlSoap.org/wsdl"
              xmlns="http://www.w3.org/2001/XMLSchema"
              xmlns:tns="http://www.ias.com/Schema/types"
              targetNameSpace="http://www.ias.com/Schema/types"
<message name="addRequest">
    <part name="a" type="xs:int">
    <part name="a" type="xs:int">
</message>
<message name="addResponse">
    <part name="a" type="xs:int">
</message>
</definitions>

```

Port type section

- This section talks about service endpoint interface and its methods.
- Port type name is the interface name and operation name is method name.
- If the interface has two methods then port type operation will be repeated for two times.
- Every operation contains I/p & O/p messages.
- When the wsdl file is given to us then 1st we need to refer to the port type section. Then next message section and then next to see schema & types section.

```

<portType name="Calculator">
    <operation name="add">
        <Input message="tns:addRequest">
        <Input message="tns:addResponse">
    </operation>
</portType>

```

Binding Section

This section following two information

- Which transfer protocol used for the communication

- Which message exchange format to be used for exchanging I/p & o/p of web service operation.

Message exchange formats are in webservice

- Style : rpc, document
 - Use : literal, encoded
1. rpc/literal
 2. rpc/encoded
 3. document/literal
 4. document/encoded

```
<binding name="calculatorBinding" type="tns:calculator">
<soap:binding transport="http://schemas.XmlSoap.org/soap/http"
style="rpc"/>
<operation name="add">
    <input>
        <soap:body use="literal">
    </input>
    <output>
        <soap:body use="literal">
    </output>
</operation>
</binding>
```

Service Section

- This section of the wsdl file tells about the address location of the web service in the network.
- It contains one or more ports and each port refers to a binding section.

```
<service name="CalculatorService">
    <port name="calculatorPort" binding="tns:calculatorBinding">
        <soap:address location="http://localhost:2022/calculator"/>
    </port>
</service>
```

Steps to download the WSDL:

- Click Setup
- Type API in Quick find/search
- Click API and Click Generate Partner WSDL
- Download WSDL

Step 1: Expose Webservice as a SOAP API.

```
global class FetchAccount1
{
    webservice static Account createAccount(String Name) {
        Account acct = new Account();
        acct.Name = Name;
        insert acct;
    }
}
```

Above code insert account and will return the inserted account.

Considerations for Using the webservice Keyword

- We can't use the webservice keyword to define a class or an inner class method, interface, or to define an interface's methods and variables.
- We can use to define top-level methods and outer class method

Step 2: Generate Partner WSDL

- Click Setup
- Type API in Quick find/search
- Click API and Click Generate Partner WSDL
- Download WSDL
-

Step 3: Convert partner WSDL into Apex class

In the source Org Convert Partner WSDL into Apex Class

Steps:-

- Click setup and type apex classes in Quick search/find
- Click generate Apex from WSDL and select the saved partner WSDL

Let us see how to consume webservice through soap API.

```

string username =''; //Target Org username
string password =''; //Target Org Password
partnerSoapSforceCom.soap sp = new partnerSoapSforceCom.soap
();
partnerSoapSforceCom.LoginResult lg =
sp.login(username,password);
AccountCreateWIthSoap.AccWebService apexweb = new
AccountCreateWIthSoap.AccWebService();
AccountCreateWIthSoap.SessionHeader_element sessionHeader =
new AccountCreateWIthSoap.SessionHeader_element();
sessionHeader.sessionId = lg.sessionId;
apexweb.timeout_x =120000;
apexweb.sessionHeader =sessionHeader;
string response = apexweb.createAccount('Anuj');
system.debug(response);

```

The above code will create the account in the target Salesforce Org.

Step 1: Create a web service method on the server-side. Create a salesforce org instance using <https://developer.salesforce.com>, which will act as a server where will create a web service method to generate the leads.

To expose SOAP services:

- Create an Apex class.
- Add Webservice method.
- Complete your server logic.
- Exposed as WSDL.

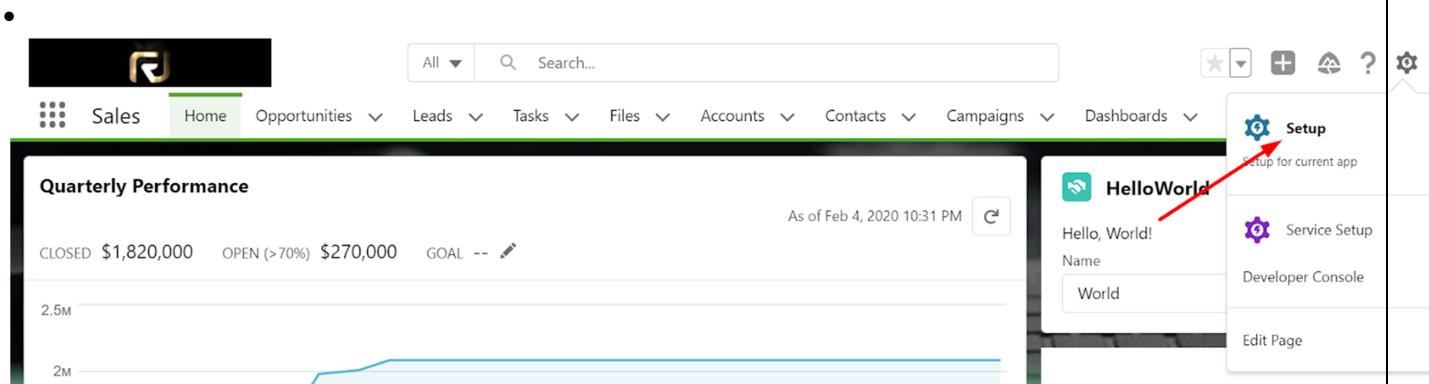
```

global class LeadManager {
    //add web service method in order to use SOAP API
    webservice static string createNewLead(String leadFirstName, String leadLastName, String leadCompanyName, String leadEmail, String leadPhone)
    {
        Lead LeadObject = new Lead();
        try{
            LeadObject.FirstName = leadFirstName;
            LeadObject.LastName = leadLastName;
            LeadObject.Company = leadCompanyName;
            LeadObject.Email = leadEmail;
            LeadObject.Phone = leadPhone;
            INSERT LeadObject;
        }catch(Exception ex){
            System.debug('Exception is:::::' +ex.getLineNumber() +' '+ ex.getMessage());
        }
        return LeadObject.Id;
    }
}

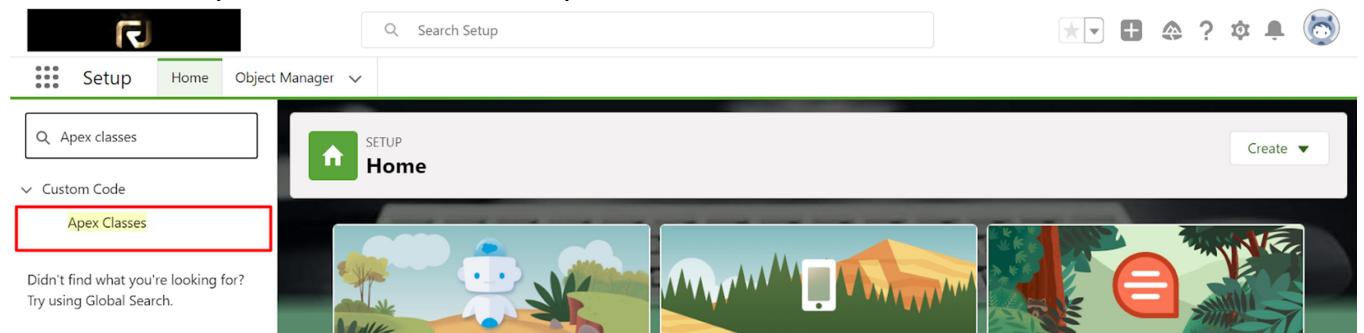
```

Step 2: Generate and share the WSDL with the client.

- Login to the salesforce -> Go to Home page -> Click on setup gear icon -> Click on setup menu.



- In the quick find box, search Apex classes.



- Search for the Apex class and click on the WSDL option to generate the WSDL file.

Apex Classes							
Edit Del Security	LeadControlHandler	48.0	Active	462	10:58 PM		
Edit Del Security	LeadController	48.0	Active	1,762	Rishabh Jain, 3/30/2020, 1:16 AM		
Edit Security	LeadConversionHandlerBatch	smagicinteract	47.0	Active	1,574	Rishabh Jain, 3/11/2021, 3:56 AM	
Edit	LeadConversionHandlerBatchTest	smagicinteract	47.0	Active	68	Rishabh Jain, 3/11/2021, 3:56 AM	
Edit Del WSDL Security	LeadManager	52.0	Active	659	Rishabh Jain, 6/15/2021, 4:42 AM		
Edit Del WSDL Security	LeadManager1	50.0	Active	660	Rishabh Jain, 6/15/2021, 4:42 AM		
Edit Del Security	LeadProcessor	48.0	Active	585	Rishabh Jain, 2/27/2020, 1:41 AM		

- Right-click on the page and save the file.

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
<!--
Web Services API : LeadManager
-->
<definitions xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://soap.sforce.com/schemas/class/LeadManager" targetNamespace="http://soap.sforce.com/schemas/class/LeadManager">
  <types>
    <xsd:schema elementFormDefault="qualified" targetNamespace="http://soap.sforce.com/schemas/class/LeadManager">
      <xsd:element name="AllowFieldTruncationHeader">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="allowFieldTruncation" type="xsd:boolean"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="CallOptions">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="client" type="xsd:string"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="DebuggingHeader">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="categories" minOccurs="0" maxOccurs="unbounded" type="tns:LogInfo"/>
            <xsd:element name="debugLevel" type="tns:LogType"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:complexType name="LogInfo">
        <xsd:sequence>
          <xsd:element name="category" type="tns:LogCategory"/>
          <xsd:element name="level" type="tns:LogCategoryLevel"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:schema>
  </types>
</definitions>
```

Right click on this file and save it.

Step 3: Client generates apex class from the WSDL.

- Login to the client Salesforce org or sign up for the new Salesforce instance that will act as a client org using <https://developer.salesforce.com/>
- Go to the Home page -> Click on the Setup gear icon -> choose the setup menu.
- Search the Apex classes in the quick find box.
- Click on the button, i.e., Generate from WSDL.
- Choose the WSDL you saved recently.

The screenshot shows the Salesforce Setup interface. The top navigation bar includes 'Setup', 'Home', and 'Object Manager'. On the left, there's a sidebar with 'Apex class' search, 'Custom Code' section, and 'Apex Classes' selected. A message says 'Didn't find what you're looking for? Try using Global Search.' Below the sidebar is a summary box: 'Percent of Apex Used: 7.32%' with a note about character usage. Under 'Apex Classes', there's a table with columns: Action, Name, Namespace Prefix, Api Version, Status, Size Without Comments, Last Modified By, Has Trace Flags, and a 'Generate from WSDL' button highlighted with a red arrow. The table contains two rows:

Action	Name	Namespace Prefix	Api Version	Status	Size Without Comments	Last Modified By	Has Trace Flags
Edit Del Security	AsyncPartnerSoapForceCom		50.0	Active	89,191	Rishabh Jain, 1/31/2021, 9:20 AM	
Edit Del Security	AsyncSOAPLeadManager		50.0	Active	2,908	Rishabh Jain, 1/12/2021, 10:42 PM	

New Apex Code from WSDL

Step 1: Choose WSDL Document

Select a WSDL document from your network file system. The WSDL document can be at most 1MB. When the document is parsed, each namespace becomes an Apex class.

Parse WSDL Cancel

Apex Classes

Apex Code is an object oriented programming language that allows developers to develop on-demand business applications on the Lightning Platform.

Percent of Apex Used: 7.32%
You are currently using 438,909 characters of Apex Code (excluding comments and @isTest annotated classes) in your organization, out of an allowed limit of 6,000,000 characters. Note that the amount in use includes both Apex Classes and Triggers defined in your organization.

Action **Name** **Namespace Prefix** **Api Version** **Status** **Size Without Comments** **Last Modified By** **Has Trace Flags**

Edit Del Security	AsyncPartnerSoapForceCom		50.0	Active	89,191	Rishabh Jain, 1/31/2021, 9:20 AM	<input type="checkbox"/>
Edit Del Security	AsyncSOAPLeadManager		50.0	Active	2,908	Rishabh Jain, 1/12/2021, 10:42 PM	<input type="checkbox"/>
Edit Del Security	faultPartnerSoapForceCom		50.0	Active	3,694	Rishabh Jain, 1/31/2021, 9:20 AM	<input type="checkbox"/>
Edit Del Security	partnerSoapForceCom		50.0	Active	10,500	Rishabh Jain, 1/31/2021, 9:20 AM	<input type="checkbox"/>
Edit Del Security	SOAPLeadManager		50.0	Active	9,850	Rishabh Jain, 1/12/2021, 4:11 AM	<input type="checkbox"/>
Edit Del Security	subjectPartnerSoapForceCom		50.0	Active	766	Rishabh Jain, 1/31/2021, 9:20 AM	<input type="checkbox"/>

Two classes will be generated: AsyncSOAPLeadManager, SOAPLeadManager

Step 4: Invoke SOAP call to the server.

- Remote site setting.
 - Settings -> Remote Site Setting

Home

Get Started with Einstein Bots
Launch an AI-powered bot to automate your digital

Mobile Publisher
Use the Mobile Publisher to create your own branded

Real-time Collaborative Docs
Transform productivity with collaborative docs.

Search Setup

Custom Code

Remote Access

Security

Remote Site Settings

Didn't find what you're looking for?
Try using Global Search.

SETUP Remote Site Settings

All Remote Sites

Below is the list of Web addresses that your organization can invoke from salesforce.com. To add another Web address, click New Remote Site.

View: All Remote Sites [Create New View](#)

Action	Remote Site Name	Namespace Prefix	Remote Site URL	Active	Created By	Created Date	Last Modified By	Last Modified Date
Edit Del	ApexDevNet	-	http://www.apexdevnet.com	✓	Jain_Rishabh	10/31/2020, 12:59 AM	Jain_Rishabh	10/31/2020, 12:59 AM
Edit Del	LeadManager	-	https://ap15.salesforce.com	✓	Jain_Rishabh	1/16/2021, 2:29 AM	Jain_Rishabh	1/16/2021, 2:29 AM

A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | Other [All](#)

New Remote Site

File ▾ Edit ▾ Debug ▾ Test ▾ Workspace ▾ Help ▾ < >

SOAPLeadManager.apxc **AsyncSOALeadManager.apxc**

Code Coverage: None API Version: 50

```

79     public Double latitude;
80     public Double longitude;
81     private String[] latitude_type_info = new String[]{"latitude",'http://soap.sforce.com/schemas/class/LeadManager',null,'1','1','false'};
82     private String[] longitude_type_info = new String[]{"longitude",'http://soap.sforce.com/schemas/class/LeadManager',null,'1','1','false'};
83     private String[] apex_schema_type_info = new String[]{"http://soap.sforce.com/schemas/class/LeadManager",'true','false'};
84     private String[] field_order_type_info = new String[]{"latitude",'longitude'};
85 }
86 public class AllowFieldTruncationHeader_element {
87     public Boolean allowFieldTruncation;
88     private String[] allowFieldTruncation_type_info = new String[]{"allowFieldTruncation",'http://soap.sforce.com/schemas/class/LeadManager',null,'1','1','false'};
89     private String[] apex_schema_type_info = new String[]{"http://soap.sforce.com/schemas/class/LeadManager",'true','false'};
90     private String[] field_order_type_info = new String[]{"allowFieldTruncation"};
91 }
92 public class LeadManager {
93     public String endpoint_x = 'https://ap15.salesforce.com/services/Soap/class/LeadManager'; Use this endpoint to create remote site setting
94     public Map<String, String> inputHttpHeaders_x;
95     public Map<String, String> outputHttpHeaders_x;
96     public String clientCertName_x;
97     public String clientCert_x;
98     public String clientCertPasswd_x;
99     public Integer timeout_x;
100    public SOAPLeadManager.DebuggingHeader_element DebuggingHeader;
101    public SOAPLeadManager.AllowFieldTruncationHeader_element AllowFieldTruncationHeader;
102    public SOAPLeadManager.DebuggingInfo_element DebuggingInfo;
103    public SOAPLeadManager.CallOptions_element CallOptions;
104    public SOAPLeadManager.SessionHeader_element SessionHeader;

```

SETUP Remote Site Settings

Remote Site Edit

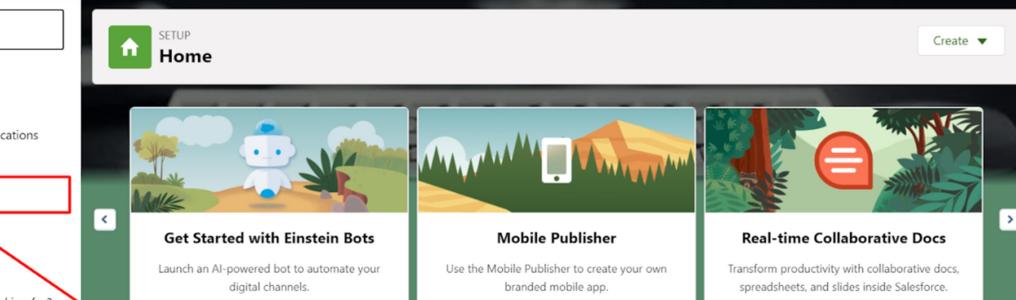
Enter the URL for the remote site. All s-controls, JavaScript OnClick commands in custom buttons, Apex, and AJAX proxy calls can access this Web address from salesforce.com.

Remote Site Edit

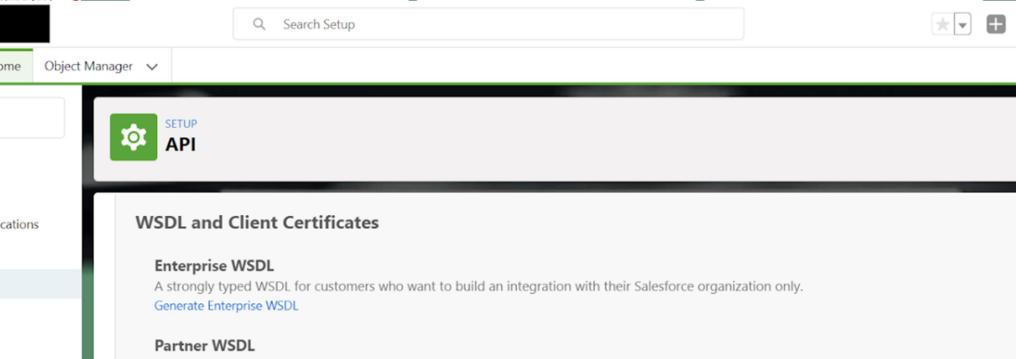
Remote Site Edit		Save	Save & New	Cancel
Remote Site Name	LeadManager			
Remote Site URL	https://ap15.salesforce.com/services/Soap/class/LeadManager			
Disable Protocol Security	<input type="checkbox"/>			
Description	<input type="text"/>			
Active	<input checked="" type="checkbox"/>			
<input type="button" value="Save"/> <input type="button" value="Save & New"/> <input type="button" value="Cancel"/>				

- Set Session – authentication parameter.

- Login to the Server-org.
 - Setup -> Quick find -> API -> Choose WSDL file.



The screenshot shows the Salesforce Home page. The left sidebar is open, displaying the 'API' section under 'Integrations'. A red box highlights the 'API' link. The main content area features three cards: 'Get Started with Einstein Bots', 'Mobile Publisher', and 'Real-time Collaborative Docs'. Each card has a small 'Create' button in the top right corner.



The screenshot shows the 'API' page under the 'Setup' tab. The left sidebar shows the 'API' section under 'Integrations' highlighted with a red box. The main content area is titled 'WSDL and Client Certificates' and contains sections for 'Enterprise WSDL', 'Partner WSDL', and 'Apex WSDL'. The 'Generate Enterprise WSDL' link in the 'Enterprise WSDL' section and the 'Generate Partner WSDL' link in the 'Partner WSDL' section are both highlighted with red boxes. A red arrow points from the 'Generate Partner WSDL' link towards the bottom of the page.

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
<!--
Salesforce.com Partner Web Services API Version 52.0
Generated on 2021-06-23 10:17:14 +0000.

Copyright 1999-2021 salesforce.com, inc.
All Rights Reserved
-->
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:fns="urn:fault.partner.soap.sforce.com" xmlns:tns="urn:partner.soap.sforce.com" xmlns:ens="urn:sobject.partner.soap.sforce.com"
    targetNamespace="urn:partner.soap.sforce.com">
    <types>
        <schema xmlns="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified" targetNamespace="urn:sobject.partner.soap.sforce.com">
            <import namespace="urn:partner.soap.sforce.com"/>
            <!-- Dynamic sObject -->
            <!-- Complex Type for sObject -->
            <complexType name="sObject">
                <sequence>
                    <element name="type" type="xsd:string"/>
                    <element name="fieldsToNull" type="xsd:string" nullable="true" minOccurs="0" maxOccurs="unbounded"/>
                    <element name="Id" type="tns:ID" nullable="true"/>
                    <any namespace="#${targetNamespace}" minOccurs="0" maxOccurs="unbounded" processContents="lax"/>
                </sequence>
            </complexType>
        </schema>
    </types>
    <!-- Complex Type for ID -->
    <schema xmlns="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified" targetNamespace="urn:partner.soap.sforce.com">
        <import namespace="urn:sobject.partner.soap.sforce.com"/>
        <!-- Our simple ID Type -->
        <!-- Simple Type for ID -->
        <simpleType name="ID">
            <restriction base="xsd:string">
                <length value="18"/>
                <pattern value="[a-zA-Z0-9]{18}"/>
            </restriction>
        </simpleType>
        <!-- Simple Type for json -->
        <simpleType name="json">
            <restriction base="xsd:string"/>
        </simpleType>
    </schema>

```

Save this file as Partner WSDL

- Edit the above WSDL with the text editor.
 - Find -> anyType and replace it with the string.

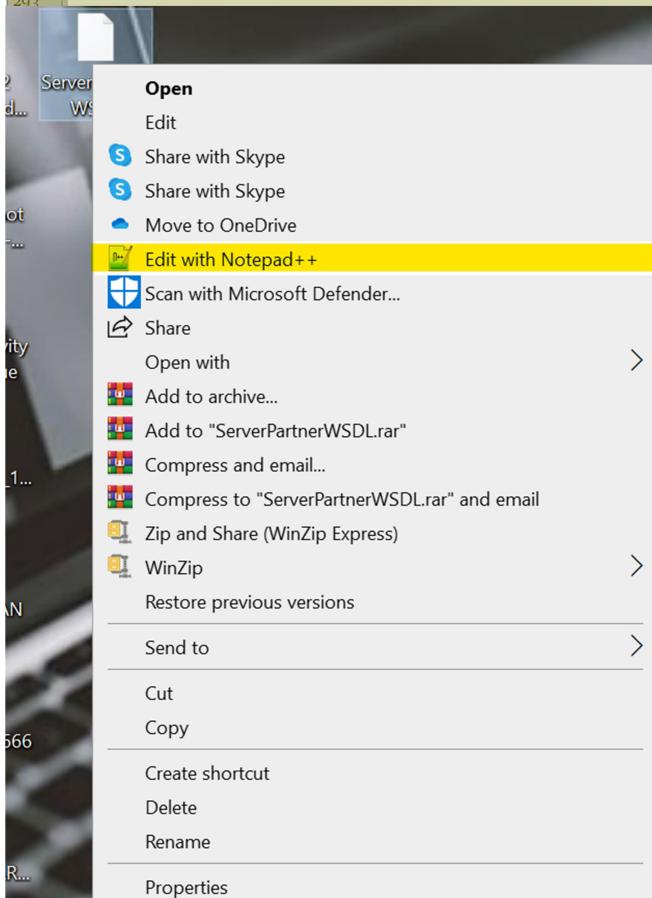
```

<complexType name="NameValuePair">
    <sequence>
        <element name="name" type="xsd:string" />
        <element name="value" type="xsd:string" />
    </sequence>
</complexType>

<complexType name="NameObjectValuePair">
    <sequence>
        <element name="isVisible" type="xsd:boolean" />
        <element name="name" type="xsd:string" />
        <element name="value" type="xsd:anyType" />
    </sequence>
</complexType>

<!-- GetUpdated Result -->
<complexType name="GetUpdatedResult">
    <sequence>
        <element name="ids" minOccurs="0" maxOccurs="unbounded" type="tns:ID" />
        <element name="latestDateCovered" type="xsd:dateTime" />
    </sequence>

```



- Go back to the Client org.

- Generate the apex class from the WSDL file generated on the Server side.

The screenshots illustrate the steps to generate Apex classes from a WSDL file:

- Step 1: Apex Classes List**
The first screenshot shows the "Apex Classes" page in the Setup menu under "Custom Code". A red box highlights the "Apex Classes" link in the sidebar. Another red box highlights the "Generate from WSDL" button in the top navigation bar.
- Step 2: Choose WSDL Document**
The second screenshot shows the "New Apex Code from WSDL" step. A red box highlights the "Choose File" button, which has the placeholder text "No file chosen".
- Step 3: Generate Apex code**
The third screenshot shows the "Step 2: Specify Class Names" step. A red box highlights the "Generate Apex code" button at the bottom right of the page.

Apex Classes Page (Top Screenshot):

New Apex Code from WSDL (Middle Screenshot):

Step 2: Specify Class Names (Bottom Screenshot):

The following generated class(es) compiled successfully with no errors:

- faultPartnerSoapForceCom
- partnerSoapForceCom
- AsyncPartnerSoapForceCom
- subjectPartnerSoapForceCom

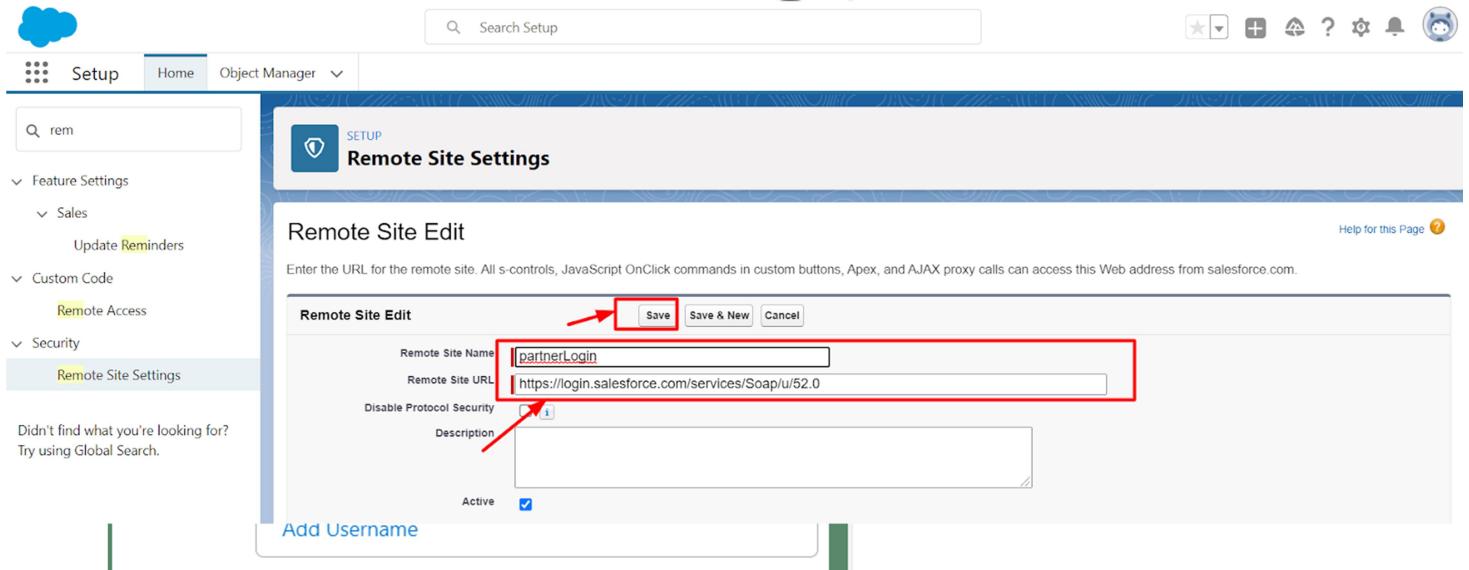
Action	Name	Namespace Prefix	Api Version	Status	Size Without Comments	Last Modified By	Has Trace Flags
Edit Del Security	AsyncPartnerSoapForceCom		50.0	Active	89,191	Rishabh Jain, 1/31/2021, 9:20 AM	<input type="checkbox"/>
Edit Del Security	SyncSOAPLeadManager		50.0	Active	2,908	Rishabh Jain, 1/12/2021, 10:42 PM	<input type="checkbox"/>
Edit Del Security	faultPartnerSoapForceCom		50.0	Active	3,694	Rishabh Jain, 1/31/2021, 9:20 AM	<input type="checkbox"/>
Edit Del Security	partnerSoapForceCom		50.0	Active	332,500	Rishabh Jain, 1/31/2021, 9:20 AM	<input type="checkbox"/>
Edit Del Security	property		52.0	Active	26	Rishabh Jain, 6/2/2021, 3:48 AM	<input type="checkbox"/>
Edit Del Security	SOAPLeadManager		50.0	Active	9,850	Rishabh Jain, 1/12/2021, 4:11 AM	<input type="checkbox"/>
Edit Del Security	subjectPartnerSoapForceCom		50.0	Active	766	Rishabh Jain, 1/31/2021, 9:20 AM	<input type="checkbox"/>

- Building Request Parameter.

- Open the generated apex class i.e partnerSoapForceCom.apxc in the client salesforce environment.
- Create a Remote site setting for the below endpoint, which will be invoked for getting the session Id.

```
public class Soap {
    public String endpoint_x = 'https://login.salesforce.com/services/Soap/u/52.0';
    public Map<String, String> inputHttpHeaders_x;
    public Map<String, String> outputHttpHeaders_x;
    public String clientCertName_x;
    public String clientCert_x;
    public String clientCertPasswd_x;
    public Integer timeout_x;
```

- Go to settings.



- Click on Reset my security token.

Sales Home Opportunities Lead

Approver Settings

Authentication Settings for External Systems

Change My Password

Connections

Grant Account Login Access

Language & Time Zone

Login History

Personal Information

Reset My Security Token

Security Central

> Display & Layout

> Email

> Chatter

> Calendar & Reminders

> Desktop Add-Ons

> Import

Reset Security Token

When you access Salesforce from an IP address that isn't trusted for your company, and you use a desktop client or the alphanumeric code that's tied to your password. Whenever your password is reset, your security token is also reset.



After you reset your token, you can't use your old token in API applications and desktop clients.

Reset Security Token

- Go to your email account and get the security token.



support@jp.salesforce.com <support@jp.salesforce.com>
to me ▾

We've sent you a new Salesforce security token because you recently changed your password or requested to reset your security token. Use desktop clients that require it.

Username: rishabh.jain@clouданология.com

Security token (case-sensitive): [m3g2e10RV7IEBxQfXDF6EtWV](#)

For more information on using your security token, see Reset Your Security Token at https://help.salesforce.com/HTViewHelpDoc?id=user_securit...

- Copy the above security token and paste it with the password in the anonymous window.

Enter Apex Code

```
1 String Username = 'Rishabh.jain@clouданология.com';
2 String Password = 'hereispasswordm3g2e10RV7IEBxQfXDF6EtWV'
```

Don't forget to Add the Security token
here in your code

Open Log Execute Execute Highlighted

- Find the Login method inside the Soap inner class and utilize that method to get the session Id from the server.

File ▾ Edit ▾ Debug ▾ Test ▾ Workspace ▾ Help ▾ < >

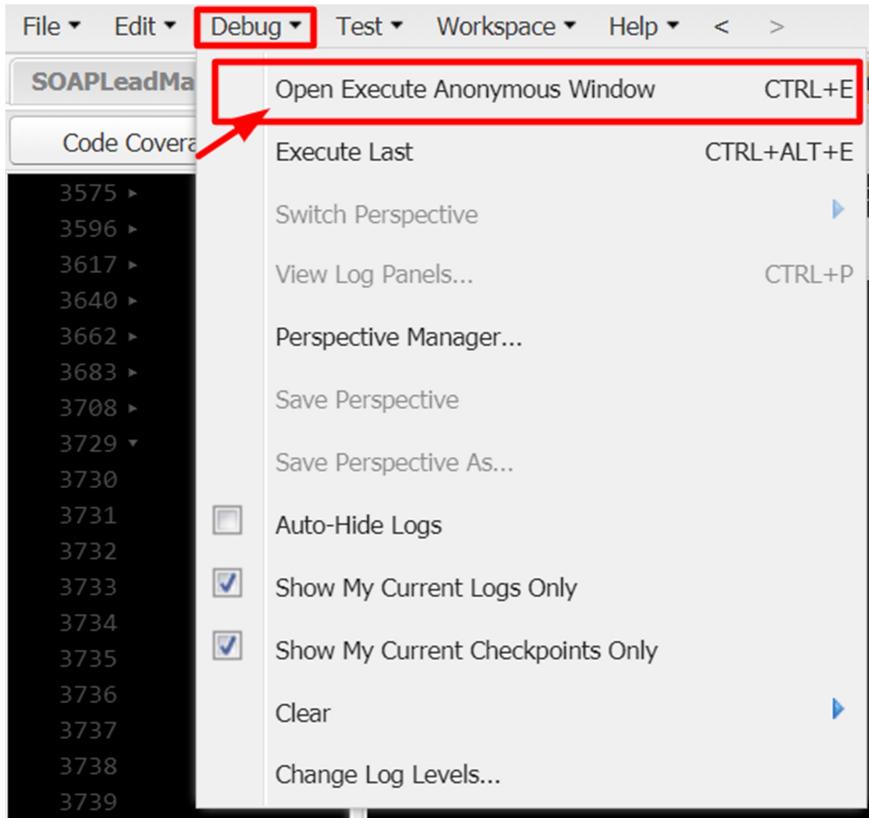
SOPALeadManager.apxc [] AsyncSOAPLeadManager.apxc [] partnerSoapSforceCom.apxc []

Code Coverage: None API Version: 52

```

3308     private String[] field_order_type_info = new String[]{"columns"};
3309 }
3310 +
3311 public class queryAllResponse_element {
3312     public partnerSoapSforceCom.QueryResult result;
3313     private String[] result_type_info = new String[]{"result", 'urn:partner.soap.sforce.com', null, '1', '1', 'false'};
3314     private String[] apex_schema_type_info = new String[]{"urn:partner.soap.sforce.com", 'true', 'false'};
3315     private String[] field_order_type_info = new String[]{"result"};
3316 }
3317 public class describeGlobalTheme_element {
3318     private String[] apex_schema_type_info = new String[]{"urn:partner.soap.sforce.com", 'true', 'false'};
3319     private String[] field_order_type_info = new String[]{};
3320 }
3321 public class Soap {
3322     public String endpoint_x = 'https://login.salesforce.com/services/Soap/u/52.0';
3323     public Map<String, String> inputHttpHeaders_x;
3324     public Map<String, String> outputHttpHeaders_x;
3325     public String clientCertName_x;
3326     public String clientCert_x;
3327     public String clientCertPasswd_x;
3328     public Integer timeout_x;
3329     public partnerSoapSforceCom.QueryOptions_element QueryOptions;
3330     public partnerSoapSforceCom.AssignmentRuleHeader_element AssignmentRuleHeader;
3331     public partnerSoapSforceCom.DisableFeedTrackingHeader_element DisableFeedTrackingHeader;
3332     public partnerSoapSforceCom.EmailHeader_element EmailHeader;
3333     public partnerSoapSforceCom.LocaleOptions_element LocaleOptions;
3334     public partnerSoapSforceCom.DebuggingHeader_element DebuggingHeader;
3335     public partnerSoapSforceCom.OwnerChangeOptions_element OwnerChangeOptions;
3336     public partnerSoapSforceCom.AllowFieldTruncationHeader_element AllowFieldTruncationHeader;
3337     public partnerSoapSforceCom.DescribeSqlListViewResult describeSqlListViewResult(String objectType, Boolean recentOnly, String isSoqlCompatibility, Integer maxRows);
3338     public partnerSoapSforceCom.DeleteResult[] delete_x(String[] ids) {#}
3339     public partnerSoapSforceCom.LoginResult login(String username, String password) {
3340         partnerSoapSforceCom.login_element request_x = new partnerSoapSforceCom.login_element();
3341         request_x.username = username;
3342         request_x.password = password;
3343         partnerSoapSforceCom.loginResponse_element response_x;
3344         Map<String, partnerSoapSforceCom.loginResponse_element> response_map_x = new Map<String, partnerSoapSforceCom.loginResponse_element>();
3345         response_map_x.put('response_x', response_x);
3346         WebServiceCallout.invoke(
3347             this,
3348             request_x,
3349             response_map_x,
3350             new String[]{endpoint_x,
3351             '',
3352             'urn:partner.soap.sforce.com',
3353             'login',
3354             'urn:partner.soap.sforce.com',
3355             'loginResponse',
3356             'partnerSoapSforceCom.loginResponse_element'});
3357     }
3358 }
```

- Open an anonymous window to execute the login method and get the session Id.



- Processing Response.

```
1 String Username = 'Rishabh.jain@clouданалоги.com';
2 String Password = 'Rish@0h120096m3g2e10RV7IEBxQfxDF6EtWV';
3
4 //call login method to login into the server
5 partnerSoapforceCom.Soap soapObj = new partnerSoapforceCom.Soap();
6 partnerSoapforceCom.LoginResult logResObj = soapObj.Login.Username,Password);
7 System.debug('log result:' + logResObj);
8 //Session header obj to get session Id
9 SOAPLeadManager.SessionHeader_element sessionObj = new SOAPLeadManager.SessionHeader_element();
10 sessionObj.SessionId = logResObj.SessionId;
11 //create lead
12 SOAPLeadManager.LeadManager leadServiceObj = new SOAPLeadManager.LeadManager();
13 //pass session Id while creating the lead
14 leadServiceObj.sessionHeader = sessionObj;
15 leadServiceObj.createNewLead('RJLead','Developer','GOOGLE','xyz@gmail.com','+917651847283');
16
17
18
```

The screenshot shows the 'Enter Apex Code' window with an Apex script for lead creation. The script uses the partnerSoapforceCom namespace to log in and create a new lead with the name 'RJLead', developer 'GOOGLE', email 'xyz@gmail.com', and phone '+917651847283'. The window includes buttons for 'Open Log', 'Execute', and 'Execute Highlighted' at the bottom.

- Execute the above script.
- Go back to the server and search for the lead you created.

The screenshot shows the Salesforce Leads page. At the top, there is a navigation bar with links for Sales, Home, Opportunities, Leads, Tasks, Files, Accounts, Contacts, Campaigns, Dashboards, Reports, Chatter, Groups, and More. A search bar contains the text "RJLead". Below the navigation bar, a header bar displays "Leads" and "Recently Viewed". The main content area shows a single lead record with the following details:

	Name	Title	Company	Phone	Mobile	Email	Lead Status	Owner Alias
1	RJLead Developer		GOOGLE	+917651847283		xyz@gmail.com	Open - Not Contacted	RJain

Red boxes highlight the "Leads" button in the navigation bar, the "RJLead" search term, and the lead record in the list.